

A Formal Model of Anonymous Systems*

Yang D. Li[†]

January 20, 2013

Abstract

We put forward a formal model of anonymous systems. And we concentrate on the anonymous failure detectors in our model. In particular, we give three examples of anonymous failure detectors and show that

- they can be used to solve the consensus problem;
- they are equivalent to their classic counterparts.

Moreover, we show some relationship among them and provide a simple classification of anonymous failure detectors.

*Some proofs are in the appendix.

[†]Email: danielly@gmail.com. Department of Computer Science and Engineering, The Chinese University of Hong Kong.)

1 Introduction

1.1 Background

The *consensus* problem [CHT96] is now recognized as one of the most important problems to solve when one has to design or to implement reliable applications on top of an unreliable asynchronous distributed system. As it is impossible to implement consensus even with one faulty process [FLP85], one of the solutions to this concern is to turn to the concept of *failure detectors*. In [CT96], the concept of unreliable failure detectors is introduced and used to solve the consensus problem in asynchronous systems. In [DGFG02], the weakest failure detector for solving consensus in the message-passing model is proved to be Ω and $\diamond W$ with a majority of correct processes. In [LH94], in the shared-memory model, Ω and $\diamond W$ are the weakest failure detector for solving consensus in any environment. The difference between [DGFG02] and [LH94] is that [LH94] has a stronger abstraction, i.e. register, in the process of implementing the consensus problem.

Further, in [DGFG10] and [DGFG⁺04], a new kind of failure detectors, Σ , is introduced, which can be used to implement register. Consequently, the weakest failure detector for solving consensus is actually (Ω, Σ) . In [DGFG02], the realistic failure detectors are considered, in [MR99], the generic protocol for solving consensus is brought forward, and in [Zie07], the eventual failure detectors are classified. Particularly, in [JT07], the fact that every problem has a weakest failure detector is shown.

[BR10] studies the failure detectors in an anonymous system, where the processes have no identity. Nevertheless, it does not provide a mathematical characterization of *anonymity*, the central concept in the paper, which results in the vagueness of the anonymous system. In this paper, we address the question of anonymity. Specifically, we provide a rigorous model for anonymous systems and show several results in our model.

1.2 A Formal Model for Anonymous Systems

We use \mathbb{N} , the set of natural numbers $\{0, 1, 2, \dots\}$, to denote the range of the clock's ticks. P means a set of n processes $\{p_1, p_2, \dots, p_n\}$. $F : \mathbb{N} \rightarrow 2^P$ is a failure pattern, i.e. the set of processes crashed at a certain time. An environment \mathcal{E} is a set of possible failure patterns. The processes can only fail by crashing (halting permanently). We assume that at least one process is correct in our model. Each process is connected to every other process via a reliable channel and message delays on these channels are unbounded but finite.

Communication can be based on the *broadcast* primitive (the same as in the classical system) and an *anonymous receive* operation (to be introduced later). We characterize the anonymity of the system by using a permutation function. Suppose that Π is a permutation function mapping from P to P . That is to say, Π is a permutation of all the processes. Let \mathcal{R} be a possibly infinite range of values that are sent to each other by the processes, R be a function mapping $P \times \mathbb{N}$ to \mathcal{R} , and $R_i(j, t)$ be the value that process i receives from another process j at time t . Intuitively, the anonymous receive operation means that the receiver cannot tell who sends out the message. Mathematically, the anonymous receive operation is defined by a R^Π function:

$$R_i^\Pi(j, t) = R_i(\Pi(j), t).$$

The system above is called an *anonymous message-passing* system. The anonymous shared-memory system is introduced in the appendix.

1.3 A Formal Definition of Anonymous Failure Detectors

Based on our formal model of anonymous systems, we introduce the formal definition of *anonymous failure detectors*. We define *crashed*(F) (faulty processes) to be $\cup_{t \in \mathbb{N}} F(t)$ and *correct*(F)

(correct processes) to be $P - crashed(F)$. Moreover, $|crashed(F)|$ means the number of faulty processes and $|correct(F)|$ represents the number of correct processes.

A failure detector history H with range V is a function from $P \times \mathbb{N}$ to V . $H(p, t)$ is the value of the failure detector module of process p at time t . A failure detector \mathcal{D} is a function that maps each failure pattern F to a set of failure detector histories $H_{\mathcal{D}}$ with range $V_{\mathcal{D}}$ (where $V_{\mathcal{D}}$ denotes the range of failure detector outputs of \mathcal{D}). $\mathcal{D}(F)$ denotes the set of possible failure detector histories permitted by \mathcal{D} for the failure pattern F . We define $F^{\Pi}(t) = \Pi(F(t))$, $\forall t \in \mathbb{N}$ and $H^{\Pi}(p, t) = H(\Pi(p), t)$, $\forall p \in P, t \in \mathbb{N}$. If $\forall \Pi, H^{\Pi} \in \mathcal{D}(F^{\Pi})$, then \mathcal{D} is called a *anonymous failure detector*.

1.4 Examples of Anonymous Failure Detectors

In this part we introduce some examples of failure detectors under our model of anonymous systems.

1.4.1 \mathcal{N}

Each failure detector module of \mathcal{N} outputs a natural number in $\{0, 1, 2, \dots, n\}$, which represents the number of processes suspected to have crashed. So the range of \mathcal{N} is $V_{\mathcal{N}} = \{0, 1, 2, \dots, n\}$.

$\mathcal{N}(F)$ is the set of all failure detector histories $H_{\mathcal{N}}$ with range $V_{\mathcal{N}}$ that satisfies the following properties:

- **Completeness** : Eventually the failure detector outputs a number that is greater than or equal to the actual number of crashed processes.
 $\exists t \in \mathbb{N}, \forall t' \in \mathbb{N} \text{ and } t' \geq t, \forall q \in P, H_{\mathcal{N}}(q, t') \geq |crashed(F)|.$
- **Accuracy** : The (correct) failure detector always outputs a number that is smaller than or equal to the actual number of crashed processes.
 $\forall t \in \mathbb{N}, \forall q \in correct(F), H_{\mathcal{N}}(q, t) \leq |crashed(F)|.$

1.4.2 $\diamond \mathcal{N}$

Each failure detector module of $\diamond \mathcal{N}$ outputs a natural number in $\{0, 1, 2, \dots, n\}$, which represents the number of processes suspected to have crashed. So the range of \mathcal{N} is $V_{\diamond \mathcal{N}} = \{0, 1, 2, \dots, n\}$.

$\diamond \mathcal{N}(F)$ is the set of all failure detector histories $H_{\diamond \mathcal{N}}$ with range $V_{\diamond \mathcal{N}}$ that satisfies the following properties:

- **Completeness** : Eventually the failure detector outputs a number that is greater than or equal to the actual number of crashed processes.
 $\exists t \in \mathbb{N}, \forall t' \in \mathbb{N} \text{ and } t' \geq t, \forall q \in P, H_{\diamond \mathcal{N}}(q, t') \geq |crashed(F)|.$
- **Eventual Accuracy** : Eventually, the output of (correct) failure detectors is a number that is smaller than or equal to the actual number of crashed processes.
 $\exists t \in \mathbb{N}, \forall t' \in \mathbb{N} \text{ and } t' > t, \forall q \in correct(F), H_{\diamond \mathcal{N}}(q, t') \leq |crashed(F)|.$

1.4.3 Θ

Each failure detector module of Θ outputs a boolean value, i.e., true or false. So under this circumstance, the range of Θ is $V_{\Theta} = \{true, false\}$.

$\Theta(F)$ is the set of all failure detector histories H_{Θ} with range V_{Θ} that satisfies the following property:

- **Eventual Self-Trust**: There is a time after which there is only one correct process, which trusts itself.
 $\exists t \in \mathbb{N}, \exists p \in correct(F), \text{ s.t. } \forall t' \in \mathbb{N} \text{ and } t' > t, H(t', p) = true, \forall q \in correct(F) - p, H_{\Theta}(t', q) = false.$

2 Consensus Algorithms

2.1 Consensus Problem

In this part we briefly review the consensus problem [CT96], which is defined by the following four properties:

- Termination: Every correct process eventually decides some value.
- Irrevocability (Integrity): Every process decides at most once.
- Agreement: No two correct processes decide differently.
- Validity: If a process decides v , then v was proposed by some process.

2.2 Consensus Algorithm with \mathcal{N}

We assume that the number of processes that may crash is bounded by f . Our \mathcal{N} based consensus algorithm proceeds in $f + 1$ asynchronous rounds. In each round every process broadcasts its value. Then it blocks until it has received enough round- r messages. In this and the following sections we assume that the message system places all received messages in a multi-set called *received*. Since we only consider messages from the round a process is currently in, we implicitly delay messages from later rounds until their round starts. For memory efficiency, messages from previous rounds can be discarded at every round switch (i.e., whenever r is increased).

Remark: Since algorithms typically wait for messages from alive processes, we deemed it more useful to use the converse of the failure detector described above. Moreover, since it is always safe to wait for messages from $n - f$ processes, we consider oracles that output the number of processes believed to be alive, denoted by \mathcal{N} and $\diamond\mathcal{N}$ respectively. In the following, we will use \mathcal{N} and $\diamond\mathcal{N}$ to denote the output of \mathcal{N} and $\diamond\mathcal{N}$ respectively.

Algorithm 1 Consensus on v with \mathcal{N}

- 1: $v \in \{0, 1\}$ initially the input value
 - 2: **for** r from 1 to $f + 1$ **do**
 - 3: broadcast (*Propose*, v, r)
 - 4: wait until received contains (*Propose*, v, r) at least \mathcal{N} times
 - 5: $values \leftarrow \{v\} \cup \{v' : (Propose, v', r) \in received\}$
 - 6: $v \leftarrow \max v' \in values$
 - 7: **end for**
 - 8: decide v
-

In this part, we show that consensus is solvable among $n \geq f + 1$ processes. To this end we start out with a lemma that shows that processes will never give up on the value 1 once they have adopted it.

Lemma 2.1 (Stubbornness). *If some correct process p adopts $v \leftarrow 1$ in some round r or p initially ($r = 0$) proposes 1, then p will have $v = 1$ for all rounds $r' > r$.*

With this intermediate step, showing Validity becomes quite simple:

Lemma 2.2 (Validity). *If a process decides v using Algorithm 1, then v was proposed by some process.*

The proof of agreement is patterned around the idea, that among the $f + 1$ rounds there must be one round during which no process crashes. We will show that this is enough for all processes to reach states such that all preferred values are equal and that once this is the case, no process can decide on another value, since no variable or message will ever carry the other value in later rounds.

Lemma 2.3. *If there exists one round, say r , which no process is in when crashing all processes will set their v to the same $\max(\text{values})$ by the end of that round, and decide on this v .*

Observing that there are $f + 1$ rounds but at most f processes can crash, it is evident that:

Observation 1. *In executions with $f + 1$ rounds, where at most f processes can crash there is at least one round in which no processes crash.*

Agreement is evident from the Lemma 2.3 and Observation 1. Validity was shown in Lemma 2.2. Irrevocability follows trivially from the algorithm, and Termination follows from the fact that the algorithm can never get stuck in a round, since the number of received messages must eventually be greater or equal to the output of \mathcal{N} in every round.

Theorem 2.4. *Algorithm 1 allows $n > f$ processes to reach Consensus.*

2.3 Consensus Algorithm with $\diamond\mathcal{N}$

We show that consensus is possible among $n > 2f$ processes when we augment our basic asynchronous model with $\diamond\mathcal{N}$.

Algorithm 2 Consensus algorithm with $\diamond\mathcal{N}$

```

1:  $v \in \{0, 1\}$  initially the input value
2:  $lock \leftarrow ?$ ,  $decided \leftarrow false$ ,  $r \leftarrow 0$ 
3: loop
4:   broadcast ( $Propose, r, v$ )
5:   wait until received contains ( $Propose, r, \_$ ) at least  $\diamond\mathcal{N}$  times
6:   proposed  $\leftarrow \{w : (Propose, r, w) \in received\}$ 
7:    $v \leftarrow \min(proposed)$ 
8:   if  $proposed - \{v\} = \emptyset$  then
9:      $lock \leftarrow v$ 
10:  else
11:     $lock \leftarrow ?$ 
12:  end if
13:  broadcast ( $Lock, r, lock, v$ )
14:  if decided then
15:    halt
16:  end if
17:  wait until received contains ( $Lock, r, \_, \_$ ) at least  $\diamond\mathcal{N}$  times
18:  locked  $\leftarrow \{w : (Lock, r, w, \_) \in received\}$ 
19:  if  $locked - \{?\} \neq \emptyset$  then
20:     $v \leftarrow \min(locked - \{?\})$ 
21:    if  $locked - v = \emptyset$  then
22:      decide  $v$ 
23:       $decided \leftarrow true$ 
24:    end if
25:  else
26:    proposed  $\leftarrow \{w : (Lock, r, \_, w) \in received\}$ 
27:     $v \leftarrow \min(proposed)$ 
28:  end if
29:   $r \leftarrow r + 1$ 
30: end loop

```

Lemma 2.5. *When some process p sends a lock message for some value, say $x \neq ?$, in round r , then no other process can send a lock message for some other value $y \notin \{x, ?\}$.*

Lemma 2.6. *In Algorithm 2, let round r be the first round where some process decides, say on x , then (1) no other process can decide a different value in round r , and (2) all other processes will decide x at most one round later.*

What remains to be shown is that there will eventually be a round in which one process is able to decide.

Lemma 2.7. *In every execution of Algorithm 2 there eventually is a round r_d where at least one process decides.*

Lemma 2.8. *Algorithm 2 guarantees Validity and Integrity.*

Theorem 2.9. *Algorithm 2 solves consensus in anonymous asynchronous systems augmented with $\diamond N$ when $n > 2f$.*

2.4 Consensus Algorithm with Θ

The difference between Ω and Θ is that with Θ only the eventual leader learns its role directly from the oracle. The most important difference is that in our algorithm only the leader sends a $(Leader, r, _)$ message. This (and that $n > 2f$) ensures that processes cannot make too much progress independently. First, however, we prove that all processes actually do make progress.

Lemma 2.10. *No correct process blocks forever in a round.*

Next we show that each process has to decide, by showing that when one or more process decides first then all other processes must decide later on, since deciding is always triggered by a $(Decide, _)$ message which processes forward before deciding. Then we prove that there is at least one first process to send that message after the leader has stabilized. Obviously the first part also holds if the $(Decide, _)$ message is sent before stabilization.

Lemma 2.11. *Every correct process decides.*

Through our final Lemma it will become evident that Agreement must hold:

Lemma 2.12. *It is impossible for $(Decide, w)$ and $(Decide, w')$ with $w \neq w'$ to be sent.*

Since processes decide on the value received via a $(Decide, v)$ message, Agreement follows from the previous Lemma, Termination from Lemma 2.11, Integrity from the fact that processes halt immediately after deciding and Validity from the fact that all values ever sent, can be easily traced back to an initial value of some process v . Therefore we have:

Theorem 2.13. *Algorithm 3 solves Consensus in asynchronous anonymous systems augmented with Θ , if $n > 2f$.*

3 Equivalence with Classical Failure Detectors

In this section we investigate the relationship between our anonymous failure detectors and the classic ones (\mathcal{P} , $\diamond\mathcal{P}$ and Ω) [CHT96] [CT96]. To this end we have to assume that unique identifiers are available and that every reception can be attributed to the sender.

Firstly, we observe that the equivalence between Ω and Θ is obvious: to obtain one from the other it is sufficient for the process that trusts itself to simply tell the other processes, or for all processes but the leader elected by Ω to simply ignore the failure detectors output. The translation of \mathcal{P} to \mathcal{N} and of $\diamond\mathcal{P}$ to $\diamond\mathcal{N}$ are obvious as well: in both cases it suffices to output the number of processes which are not suspected.

The remaining relations are explored in more detail via transformations. By \mathcal{DF} we denote the asynchronous algorithm that implements \mathcal{F} based on \mathcal{D} . Both transformations work by

Algorithm 3 Consensus algorithm with Θ

```
1:  $v \in \{0, 1\}$  initially the input value
2:  $r \leftarrow 0$ 
3: Code for processes  $p$ :
4: loop
5:   wait until  $\Theta = \text{true}$  or received contains  $(\text{Leader}, r, \_)$  at least once
6:   if received $(\text{Leader}, r, w)$  then
7:      $v \leftarrow w$ 
8:   else
9:     if  $\Theta = \text{true}$  then
10:      broadcast $(\text{Leader}, r, v)$ 
11:    end if
12:  end if
13:  broadcast $(\text{Report}, r, v)$ 
14:  wait until received contains  $(\text{Report}, r, \_)$  at least  $(n - f)$  times
15:  if  $\exists w : \text{received}(\text{Report}, r, \_)$  from  $> n/2$  processes then
16:     $aux \leftarrow w$ 
17:  else
18:     $aux \leftarrow ?$ 
19:  end if
20:  broadcast $(\text{Vote}, r, aux)$ 
21:  wait until received contains  $(\text{Vote}, r, \_)$  at least  $(n - f)$  times
22:  if received $(\text{Vote}, r, aux')$  with  $aux' \neq ?$  then
23:     $v \leftarrow aux'$ 
24:  end if
25:  if received $(\text{Vote}, r, aux')$  with  $aux' \neq ?$  at least  $n - f$  times then
26:    broadcast $(\text{Decide}, v)$ 
27:  end if
28:   $r \leftarrow r + 1$ 
29: end loop
30: upon reception of  $(\text{Decide}, v)$  do
31:  broadcast $(\text{Decide}, v)$ 
32:  decide decision
33: halt
```

building a estimate of the alive processes, denoted by AL , and then suspecting all processes that are not in this set, i.e., $P - AL$, where P denotes the set of all processes.

Algorithm 4 $\diamond\mathcal{N} \diamond \mathcal{P}$ Implementation

```

1: Code for processes  $p$ :
2:  $r \leftarrow 0$ 
3:  $suspect \leftarrow \emptyset$ 
4: loop
5:    $r \leftarrow r + 1$ 
6:   broadcast  $(ALIVE, r)$ 
7:   wait until received  $\diamond\mathcal{N}(ALIVE, r)$  messages from the set  $AL$ 
8:    $suspect \leftarrow P - AL$ 
9: end loop

```

The implementation of $\diamond\mathcal{N} \diamond \mathcal{P}$ (Algorithm 4) is quite simple. Since only eventual Strong Accuracy is required, it suffices to output those processes that did not send $(ALIVE, _)$ messages in the current round. Thus wrong suspicions can only occur in rounds where the crashed processes have sent messages before crashing, and these messages are faster than those from alive processes. In some later round, this wrong is [eventually] corrected, due to the absence of a message from the crashed process.

Let us now reiterate the properties of \mathcal{P} and $\diamond\mathcal{P}$:

- Strong Completeness. Eventually every process that crashes is permanently suspected by every correct process.
- Strong Accuracy. No process is suspected before it crashes.
- Eventual Strong Accuracy. There is a time after which correct processes are not suspected by any correct process.

Theorem 3.1. *The implementation $\diamond\mathcal{N} \diamond \mathcal{P}$ (Algorithm 4) guarantees Strong Completeness and Eventual Strong Accuracy.*

Algorithm 5 $\mathcal{N}\mathcal{P}$ Implementation

```

1: Code for processes  $p$ :
2:  $r \leftarrow 0$ 
3:  $suspect \leftarrow \emptyset$ ;  $earlierlive \leftarrow \emptyset$ ;  $lastchange \leftarrow 0$ 
4: loop
5:   broadcast  $(ALIVE, r)$ 
6:   wait until received  $(ALIVE, r)$  messages from some set  $AL$  and  $|AL| = \mathcal{N}$ 
7:   if  $AL \neq earlierlive$  then
8:      $lastchange \leftarrow r$ 
9:   else
10:    if  $r \geq lastchange + f + 2$  then
11:       $suspect \leftarrow P - AL$ 
12:    end if
13:  end if
14:   $earlierlive \leftarrow AL$ 
15:   $r = r + 1$ 
16: end loop

```

Since \mathcal{P} is not allowed to make wrong suspicions, we have to make sure that AL always contains all processes that have not crashed whenever we update suspect. To ensure this we wait until this set does not change for $f+2$ rounds. Before we turn to proving that our translation guarantees Strong Accuracy and Strong Completeness, we show that all alive processes proceed through the rounds in a somewhat coordinated way:

Lemma 3.2. *At any time, the difference between the round numbers of two alive processes p and q is smaller than or equal to $f + 1$.*

Lemma 3.3. *The Translation \mathcal{NP} guarantees Strong Accuracy.*

Lemma 3.4. *The Translation \mathcal{NP} guarantees Strong Completeness.*

From the lemmas above it follows immediately that

Theorem 3.5. *The Translation \mathcal{NP} guarantees Strong Completeness and Strong Accuracy.*

4 A Simple Classification and Reductions

4.1 A Formal Model for Reductions among Anonymous Failure Detectors

We say an anonymous failure detector \mathcal{D}' can be reduced to another failure detector \mathcal{D} (\mathcal{D} is stronger than \mathcal{D}') if there is an algorithm $T_{\mathcal{D} \rightarrow \mathcal{D}'}$ that transforms \mathcal{D} to \mathcal{D}' under an environment \mathcal{E} . $T_{\mathcal{D} \rightarrow \mathcal{D}'}$ (using \mathcal{D}) maintains a variable $output_p$ at every process p , which emulates the output of \mathcal{D}' at p . Let O be the history of all the output variables and we require $O^\Pi \in \mathcal{D}'(F^\Pi)$, where $F \in \mathcal{E}$ and Π is a permutation function of all the processes.

4.2 A Simple Classification

In short, \mathcal{N} and $\diamond\mathcal{N}$ belong to the same class of failure detectors, the *symmetric* failure detectors. Θ is another class of failure detectors, the *unsymmetrical* failure detectors. The difference of the two kinds of failure detectors is that the symmetric failure detector outputs the same information at all correct processes while the unsymmetrical failure detectors do not.

This is a very simple classification. In an anonymous system, a process cannot distinguish other processes and only knows itself. So usually unsymmetrical failure detectors output one value at a specific process and output some other value at the rest of processes.

4.3 Reductions among Anonymous Failure Detectors

Now let's talk about the relations of the anonymous failure detectors mentioned above.

Theorem 4.1. *\mathcal{N} is stronger than $\diamond\mathcal{N}$.*

Proof. They both have the property of completeness. The accuracy property of \mathcal{N} clearly implies the eventual property of $\diamond\mathcal{N}$ while the reverse is not true. \square

Theorem 4.2. *\mathcal{N} and Θ are incomparable.*

Proof. Obviously, there is no deterministic reduction from Θ to \mathcal{N} as there is simply no way to break the symmetry in \mathcal{N} . Further, there is no deterministic reduction from \mathcal{N} to Θ . By contradiction, assume that there exists a reduction algorithm A such that for each failure pattern F and failure detector history $H \in \Theta(F)$, A outputs a failure detector history $H' \in \mathcal{N}(F)$. Denote q to be a correct process. Since \mathcal{N} can be implemented, there should exist $t_0 \in \mathcal{N}$ such that after t_0 , i.e. for $t > t_0$, $H' \in \mathcal{N}(F)$ should satisfy completeness and accuracy, i.e. $H_{\mathcal{N}}(q, t) = |crashed(F)|$. Without the loss of generality, suppose that the only process that trusts itself in the Θ output is process p_1 . Then we let p_1 be silent until $t_1 > t_0$, then at time t_2 that $t_0 < t_2 < t_1$, the run of the algorithm cannot distinguish the circumstance that p_1 is slow

from the one that p_1 is dead. Therefore, $H_{\mathcal{N}}(q, t_2) > |crashed(F)|$, which violates the property of accuracy in the definition of \mathcal{N} . □

Theorem 4.3. $\diamond\mathcal{N}$ and Θ are incomparable.

Proof. Following a similar argument in the proof of the previous theorem, it is easy to show that $\diamond\mathcal{N}$ cannot be reduced to Θ . Also, there is no reduction from Θ to $\diamond\mathcal{N}$ due to symmetry reasons. □

4.4 Randomized Reductions

The reductions discussed above are *deterministic* reductions. We also define the notion of *randomized* reduction. Instead of requiring $O^\Pi \in \mathcal{D}'(F^\Pi)$, we allow the use of randomness and only require $O^\Pi \in \mathcal{D}'(F^\Pi)$ to be correct with probability at least $2/3$. We show an example of randomized reductions.

Theorem 4.4. Under randomized reduction, \mathcal{N} is stronger than Θ .

Proof. We will show a reduction algorithm converting \mathcal{N} to Θ . The converse is not possible due to the proof of Theorem 4.2.

At first, for each process p_i , it randomly generates a real number. In theory, the number of real numbers are infinite and so the chance for two processes that get the same real number is 0. However, in practice, it may be hard to generate an infinite number of numbers. So here the pool may be finite and there always exists the probability that two processes get the same real number. However, if we let the pool to be large enough, much larger than the number of processes, then the probability for two processes to get the same real number is extremely low and will tend to 0 if the pool is going to infinity.

This is where the randomness lies. Then we can assume that each process has a distinct real number. In the following process, when it tries to broadcast, it should include this real number. You may think that we have return to the situation of the classic systems. In some sense this thinking is right and some other sense, it is not. Although p_1 receives distinct real numbers, it cannot tell whether a real number it receives, is from p_2, p_3, \dots , or p_n . Thus this is consistent with the definition of the anonymous system model.

Then why do we say that in some sense we just return to the classic systems? This can be attributed to the anonymous model. The anonymity is that our real numbers are a permutation of the process id's, in which sense we cannot distinguish the processes. Nonetheless, if we treat the real numbers as the identifiers, then the anonymity just disappears. Then we can comfortably utilize the CHT proof to extract a certain real number corresponding to a certain process that we do not know. Each process at this stage can judge if the extracted real number is its own initial value. If the answer to this is affirmative, then this process is just the process we seek in the Θ failure detector. □

5 Concluding Remarks

In summary, we provide a rigorous model for anonymous systems and discuss some issues related to failure detectors under our model. We hope that further problems and notions can be brought forward in our model. For instance, more examples of failure detectors can be shown. Moreover, we believe that the major open problem in the anonymous system is the weakest anonymous failure detector for consensus. It may be hard to know the weakest anonymous failure detector in a deterministic sense; so we have defined randomized reduction. We hope that the weakest anonymous failure detector for consensus under randomized reduction can be easier.

References

- [BR10] Francois Bonnet and Michel Raynal. Anonymous asynchronous systems: The case of failures detectors. *Proceedings of the 24th International Symposium on Distributed Computing*, pages 206–220, 2010.
- [CHT96] Tushar Chandra, Vassos Hadzilacos, and Sam Toueg. The weakest failure detector for solving consensus. *Journal of the ACM*, 43(4):685–722, 1996.
- [CT96] Tushar Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, 1996.
- [DGFG02] Carole Delporte-Gallet, Hugues Fauconnier, and Rachid Guerraoui. A realistic look at failure detectors. *Proceedings of the 2002 International Conference on Dependable Systems and Networks*, pages 345–352, 2002.
- [DGFG⁺04] Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui, Vassos Hadzilacos, Petr Kouznetsov, and Sam Toueg. The weakest failure detectors to solve certain fundamental problems in distributed computing. *Proceedings of the 23rd annual ACM symposium on Principles of distributed computing*, pages 338–346, 2004.
- [DGFG10] Carole Delporte-Gallet, Hugues Fauconnier, and Rachid Guerraoui. Tight failure detection bounds on atomic object implementations. *Journal of the ACM*, 57(4):Artical 22, 2010.
- [FLP85] Michael Fischer, Nancy Lynch, and Michael Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, 1985.
- [JT07] Prasad Jayanti and Sam Toueg. Every problem has a weakest failure detector. *Proceedings of the 27th ACM symposium on Principles of distributed computing*, pages 75–84, 2007.
- [LH94] Wai-Kau Lo and Vassos Hadzilacos. Using failure detectors to solve consensus in asynchronous shared-memory systems. *Proceedings of the 8th International Workshop on Distributed Algorithms*, pages 280–295, 1994.
- [MR99] Achour Mostefaoui and Michel Raynal. Solving consensus using Chandra-Toueg’s unreliable failure detectors: A general quorum-based approach. *Proceedings of the 13th International Symposium on Distributed Computing*, pages 49–63, 1999.
- [Zie07] Piotr Zielinski. Automatic classification of eventual failure detectors. *Proceedings of the 21st International Symposium on Distributed Computing*, pages 465–479, 2007.

A A Formal Model of Anonymous Shared-Memory System

As in the anonymous message-passing system, there is a set of processes $P = \{p_1, p_2, \dots, p_n\}$ in the anonymous shared-memory system. In addition, there are m objects $O = \{o_1, o_2, \dots, o_m\}$. We assume that each process has a (possibly infinite) state machine and a set of states, one of which is the initial state. Each state q of process p has three special fields:

- $q.obj$, the object to be accessed next, or null
- $q.op$, the operation on $q.obj$ to be executed
- $q.in$, the input parameter (if any) of $q.op$

We use a permutation function Π , which maps P to P . The configuration C of the system is the states of all processes and the values of all shared objects, i.e. vector $(q_1, q_2, \dots, q_n, v_1, v_2, \dots, v_m)$. We define q_i to be the state of $\Pi(p_i)$, for $i = 1, 2, \dots, n$, and v_m to be the value of o_m , for $m = 1, 2, \dots, m$. The function f is the state transition machine from some state and value (q, v) to some state q' .

Therefore, just like what we did in the anonymous message-passing model, we also use permutation to characterize the anonymity in the anonymous shared-memory system.

B Proofs for Algorithm 1

B.1 Proof of Lemma 2.1

A process p always adds its current value to values in line 5, and always chooses the maximum of all values in line 6, therefore once 1 was adopted it will always remain the maximum (since 0 and 1 are the only possible values) and the Lemma follows.

B.2 Proof of Lemma 2.2

Since we are considering binary consensus only, there are only two cases where Validity could be violated: (1) Either some process p decided 1 when all processes had 0 as their initial value, or (2) some process p decided 0 and all processes had 1 as their initial value. In case (1) p must have received 1 from some other process at some point, otherwise 1 cannot become a member of values, and p initially proposes 0 by assumption. Since all processes only send their current estimate v , some process must have initially proposed 1, which is a contradiction to the assumption of (1). The impossibility of case (2) follows from Lemma 2.1.

B.3 Proof of Lemma 2.3

Let f_{r-1} denote the number of processes that have crashed up to and including round $r - 1$. When no processes crash during round r , the processes will wait for messages from $n - f_{r-1}$ processes, since \mathcal{N} will never output a number smaller than the number of alive processes. This, however, implies that all processes get the same set of messages in round r , and thus they all have the same set of values in their respective values sets. Therefore the maximum of round r , denoted m_r , will be the same at all alive processes. Now agreement follows from the fact that no process can send a value $v \neq m_r$ in rounds $r' > r$, and therefore $\forall r' > r : m_{r'} = m_r$. Thus all processes will decide on m_r .

C Proofs for Algorithm 2

C.1 Proof of Lemma 2.5

Since p sends $(Lock, r, x, x)$ it cannot have received any propose messages for any other value (otherwise it could not have reached line 9). Since processes wait for $\diamond\mathcal{N} \geq n - f > n/2$ messages, every other process must have received at least one $(Propose, r, x)$ message, keeping them from reaching line 9 with $v \neq x$. Thus each processes either sends out a lock message for x or $(Lock, r, ?, -)$.

C.2 Proof of Lemma 2.6

Let p denote the deciding process; to be able to decide, p requires $\diamond\mathcal{N}(Lock, r, x, -)$ messages. From Lemma 2.5, it follows that no process q can have received sufficiently many messages to decide another value in this round. Therefore, (1) holds.

Since p process has received $\diamond\mathcal{N} \geq n - f \geq f + 1$ lock messages, any other process must have received at least one of these and thus all reach line 20 and calculate the same minimum, i.e., the only value x , and use this value as input for the next round. Since all alive processes now propose the same value in the proposal phase of round $r + 1$, all processes receive $\diamond\mathcal{N}$ messages containing x . This in turn results in all alive processes to lock x and send enough lock messages to force all processed (that did not decide in r and therefore terminated after broadcasting their lock messages) to decide in round $r + 1$, thereby ensuring (2).

C.3 Proof of Lemma 2.7

For the sake of contradiction assume otherwise, and let r_a denote the first round where the all alive processes $\diamond\mathcal{N}$ is accurate at the start of the round (line 4). Moreover, let r_c denote the first round after the round in which the last process crashed. Since we assume that no process ever decides, both of these rounds must exist. Let $r_d = \max\{r_a, r_c\}$, then all processes will receive all the messages from all alive processes, in all rounds $r \geq r_d$. Therefore all must calculate the same minimum, say x , in line 7 from r_d on. If all values received were the same, this value is also locked by all processes and therefore decided on in the second phase, leading to a contradiction, since there is a decision. So assume that there where different values in the propose messages, and thus $proposed - x \neq \emptyset$; in line 8, which results in locked to contain ? in line 19, i.e., no decision is possible in this round. However, all processes set $v \leftarrow x$ in line 27. Now all processes have the same input value at the start of round $r_d + 1$, which leads to all processes to decide x by the argument for (2) in Lemma 2.6.

C.4 Proof of Lemma 2.8

Since no process ever sets v to a value that is neither its own initial value nor a value received from another process, it follows trivially that when v is decided on, this value must be some processs input value, thereby ensuring Validity. Finally, Integrity follows from halting in line 15 in the round after deciding.

D Proofs for Algorithm 3

D.1 Proof of Lemma 2.10

The proof is by contradiction. Let r be the smallest round in which a process blocks forever. Blocking can only occur at one of the three wait statements. We will show that it is impossible to wait forever for each of them.

The first wait, requires that in phase r , no process will ever trust itself to be the leader, and thus not unblock itself directly and all other correct processes via a $(Leader, r, -)$ message, contradicting the properties of Θ .

Since no correct process can block forever at the first wait, all correct processes will eventually send a $(Report, r, _)$ message thus unblocking all processes in the second wait (since at most f processes can crash and thus not send a $(Report, r, _)$ message). The same is true about the third wait statement and $(Vote, r, _)$ messages.

Therefore, no correct process was blocked forever in each of the three waits and thus they will all start round $r + 1$, which contradicts the assumption that some processes will block in round r .

D.2 Proof of Lemma 2.11

When one process decides, it has successfully broadcast a $(Decide, v)$ message in the line before. This message will eventually arrive at every other process and cause it to decide (if it did not decide before). Thus, when one process decides, every correct process decides.

We now prove that at least one process sends a $(Decide, v)$ message. Due to Lemma 2.10 and the properties of Θ , eventually there is a round in which only one process sends a $(Leader, r, v)$ message, since this message is eventually received by all correct processes. They will all set their v to the same value w , and thus $n - f$ identical messages will be sent and received in the second phase and thus $aux = w$ at all alive processes. And thus every alive process will send a $(Decide, v)$ message (with the $v = w$).

D.3 Proof of Lemma 2.12

We show that when one of the two messages (w.l.o.g. $(Decide, w)$) is sent in round r , then (1) the other cannot be sent in r or in later rounds, and (2) all processes have $v = w$.

First we note that in some given round aux cannot take two different non-? values, as only one value can reach a majority in a benign system. As the value sent out via $(Decide, w)$ messages cannot be ?, it is clear that $w = aux$. Thus all $(Decide, _)$ messages sent in r must contain the same value.

Secondly, we observe that v contains the same variable at all correct processes at the end of round r since one process sending out a $n - f$ message in line 26 implies that every other process will receive at least one $(Vote, r, aux')$ with $aux' \neq ?$ since $n > 2f$. From this it follows that no other value can ever be decided after round r (no other value will ever be proposed by a leader).

E Proof of Theorem 3.1

Quite simply, eventually $\diamond \mathcal{N}$ will return the correct number of processes that have not crashed, let this time be denoted by t' . Eventually all messages from crashed processes have been delivered at all processes, let this time be denoted by t'' . Let further $t = \max(t', t'')$ and r_{max} the maximum round at time t . After t we know that the output of $\diamond \mathcal{N}$ is accurate, thus all processes can only succeed to set their round to some value $r > r_{max} + 1$, if they have received $(Alive, r - 1)$ messages from all alive processes (which are the correct processes). These processes can be found in AL at each correct process. Therefore from this time on, the set of suspected processes we have $suspect = P - AL = SF$, where SF denotes the set of processes that have crashed. In other words p will permanently suspect all crashed processes. Thus we have shown that $\diamond \mathcal{N} \diamond \mathcal{P}$ implements Strong Completeness. Moreover, since no correct process is suspected in any round that starts after t it follows that Eventual Strong Accuracy is guaranteed as well.

F Proofs for Algorithm 5

F.1 Proof of Lemma 3.2

Without loss of generality assume that round number of q is further advanced than the round number of p . We now assume by contradiction that q has reached round $r_p + f + 2$ (with r_p denoting p 's round number). Obviously q did not receive a round r' message from p for any $r' > r_p$. The only way for q to pass the wait statement in line 6 is for some other process to send its round r' message and then crash, thus eventually decreasing the output of \mathcal{N} at q . Since q has reached round $r_p + f + 2$ this must have happened $f + 1$ times, which contradicts the definition of f as the maximum number of failures in any execution.

F.2 Proof of Lemma 3.3

Assume by contradiction that some process p is put into the suspect set of some process q before it crashes. This requires that p is not in the alive processes set AL of q for $f + 2$ rounds. Moreover for p to turn up in q 's suspect set the contents of AL cannot have changed for $f + 2$ rounds. This also implies that \mathcal{N} did not change for $f + 2$ rounds. Due to the accuracy property of \mathcal{N} (it never outputs a number smaller than the number of alive processes) this in turn implies that no process crashed while q performed the previous $f + 2$ rounds. Due to Lemma 3.2 no process that crashed before can have sent messages for all these rounds. Thus, p must be in AL contradicting the assumption that it was not.

F.3 Proof of Lemma 3.4

Since a faulty process only takes finitely many steps, it can only send messages for finitely many rounds. Therefore, there exists a round from which on no process receives messages from it, thus it is not part of the set AL (the alive processes estimate) at any process. This results in crashed processes to be eventually suspected.